

The Ultimate Guide to Semantic Reasoning

How to enrich your data for practical applications including use with RAG & LLMs

Tom Vout

Knowledge Engineer

Tom.vout@oxfordsemantic.tech



What can Reasoning do for you?



```
1
2 PREFIX : <http://www.oxfordsemantic.tech/f1demo/>
3
4 # Drivers with their win percentage, ordered by win percentage
5 SELECT ?forename ?surname ?raceCount ?raceWins ?percentage
6 WHERE {
7   # First get the drivers
8   ?driver :driver_forename ?forename ;
9           :driver_surname ?surname .
10
11 # Then get the race count for each driver with an innery query...
12   {SELECT ?driver (COUNT(?race) AS ?raceCount)
13     WHERE {
14       ?result :result_driver ?driver ;
15              :result_race ?race .
16     }
17   GROUP BY ?driver}
18
19 # ... and get the *win* count for each driver with another inner query.
20   {SELECT ?driver (COUNT(?race) AS ?raceWins)
21     WHERE {
22       ?result :result_driver ?driver ;
23              :result_race ?race ;
24              :result_positionOrder 1 .
25     }
26   GROUP BY ?driver}
27
28 # Finally use the two aggregate variables to compute a percentage
29 # with the BIND keyword.
30 BIND(?raceWins/?raceCount AS ?percentage)
31
32 }
33 ORDER BY DESC(?percentage)
```

Fetches 108 answers in 0.011 s. ✓

**Datalog
rules**

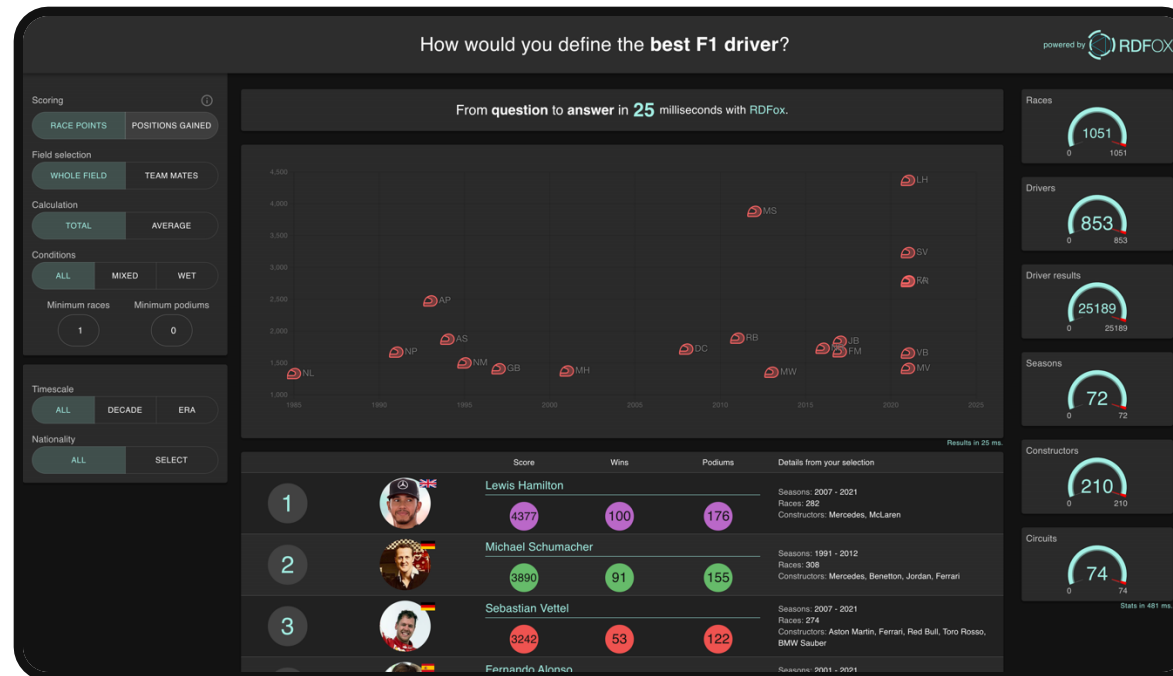
```
1
2 PREFIX : <http://www.oxfordsemantic.tech/f1demo/>
3
4 SELECT ?forename ?surname ?percentage
5 WHERE {
6   ?driver :driver_forename ?forename;
7           :driver_surname ?surname;
8           :hasWinPercentage ?percentage;
9           :hasRaceCount ?count .
10 }
11 ORDER BY DESC(?percentage)
```

Fetches 108 answers in 0.003 s. ✓

F1 Dashboard with Knowledge Graphs



Controls and filters for the scoring system.



Statistics about the data used to form the results.

This matches the filters.

Try it for yourself!

<http://f1.rdfox.tech>

Follow Along



Scan this with
your phone!
You will receive an
email with the
workshop package.

A.

Download the CDL 25 RDFox workshop package

[Download the package here](https://www.oxfordsemantic.tech/signup/cdl25)

www.oxfordsemantic.tech/signup/cdl25

1. Complete the form.
2. Wait to receive an email with a link to the package.
3. Download & unzip the package on your computer.
4. Open the 'RDFox at CDL 25' PDF.
5. Follow the instructions on slide 10 onwards.



Or search for 'VS
Code' in your
browser.

B.

Download VS Code

[Download VS Code here](https://code.visualstudio.com/)

<https://code.visualstudio.com/>

Or your IDE of choice
(setup steps won't be covered)

Objectives



The Foundations

- What is Semantic Reasoning?
- What is RDFox?
- Setting up RDFox
- Querying with SPARQL

OWL Reasoning

- Why do we need reasoning?
- Sub & Inverse Properties
- Class Equivalence
- Unions & Intersections
- Limitations of OWL

Datalog Reasoning

- Basic rules
- Filters
- Aggregates
- Negation
- Binds
- Incremental Reasoning
- Materialization vs. query rewriting
- RDFox in enterprise

Please feel free to ask questions at any time! Exercises are scattered throughout. Links are provided for reference.

<https://docs.oxfordsemantic.tech/>

Objectives



The Foundations

- What is Semantic Reasoning?
- What is RDFox?
- Setting up RDFox
- Querying with SPARQL

OWL Reasoning

- Why do we need reasoning?
- Sub & Inverse Properties
- Class Equivalence
- Unions & Intersections
- Limitations of OWL

Datalog Reasoning

- Basic rules
- Filters
- Aggregates
- Negation
- Binds
- Incremental Reasoning
- Materialization vs. query rewriting
- RDFox in enterprise

What is Semantic Reasoning?



Semantic Reasoning (rules-based AI) enriches a Knowledge Graph by adding new information



This combines knowledge and data to answer more complex questions beyond queries alone



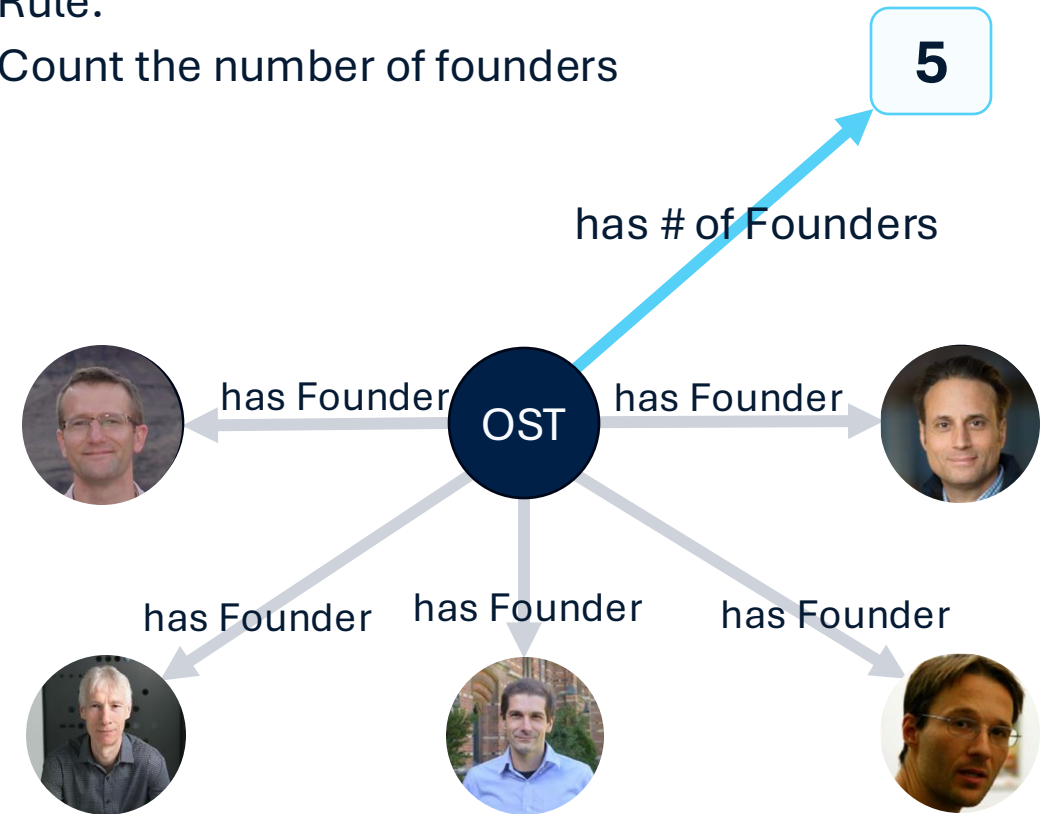
It drastically improves performance of queries by doing the hard work ahead of time



Incremental Reasoning updates automatically with new data

Rule:

Count the number of founders



What is RDFox[®]?



RDFox is a Knowledge Graph database
and Semantic Reasoning Engine



**Incremental
Semantic
Reasoning**



**Speed &
Scalability**



On Device



**Oxford
University**



Setting up RDFox

- Download the workshop material
- Start RDFox
- Load the data



Download the appropriate package




- 1 Download the package that corresponds to your operating system



[Download the package here](https://www.oxfordsemantic.tech/signup/cdl25)

www.oxfordsemantic.tech/signup/cdl25

1. Fill out the form
2. You will receive an email with a link to download the package



CDL 2025 Workshop Package

The Ultimate Guide to Semantic Reasoning & Knowledge-based AI

How to enrich your data for practical applications including use with RAG & LLMs

Download the appropriate workshop package for your operating system:

MAC (INTEL)	MAC (ARM)	LINUX (x86)	LINUX (ARM)	WINDOWS
macOS 10.14 or higher	macOS 11 or higher	Ubuntu 18.04, Amazon Linux 2, or higher	Ubuntu 18.04, Amazon Linux 2, or higher	Windows 10 or higher

- 2 Unzip the package

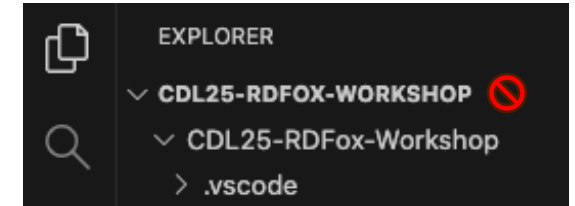
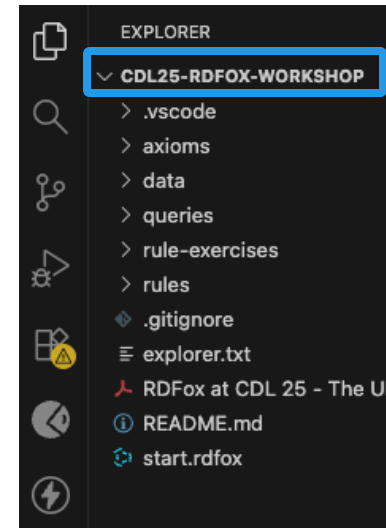
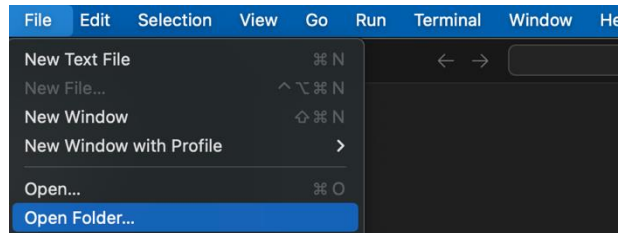
Opening the terminal in VS Code



1 Open VS Code

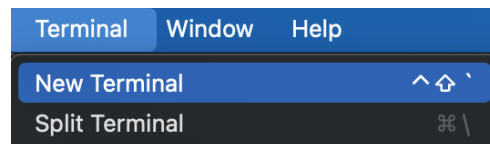
2 Open the workshop folder with **File > Open Folder**

Select the folder directly containing the 'data', 'rules', 'queries' folders etc.

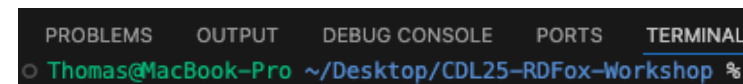


If the folder name appears twice, repeat step 2 and open the inner folder

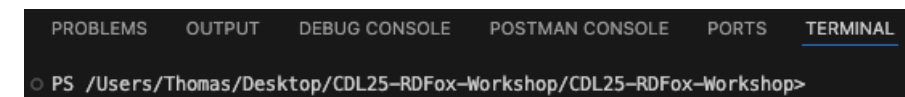
3 Open a new terminal with **Terminal > New Terminal**



› E.g. MacOS (zsh)



› E.g. Windows (PowerShell)



This path is unique to you

Starting RDFox with a script



- 1 In the terminal, from the 'RDFox Workshop' working directory, run the relevant command for the version of RDFox you downloaded:

You will need to edit the command if you are not using RDFox v7.4b or are running on LINUX

› MacOS ARM

```
./RDFox-macOS-arm64-7.4b/RDFox sandbox . start
```

› MacOS INTEL

```
./RDFox-macOS-x86_64-7.4b/RDFox sandbox . start
```

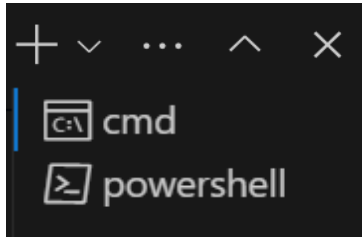
The Windows command depends on the shell used – shown right of the terminal window

› Windows (CMD)

```
RDFox-win64-x86_64-7.4b\RDFox.exe sandbox . start
```

› Windows (PowerShell)

```
& RDFox-win64-x86_64-7.4b\RDFox sandbox . start
```



Be Aware! When copying and pasting these commands into the terminal may change the formatting including spacing and special characters.

Please check that they are correct.



Starting RDFox with a script

2 The RDFox server should now be running!

```
Thomas@MacBook-Pro SEMANTICS2024RDFoxWorkshop % rdfbox sandbox . start
Source code for RDOx v1.0 Copyright 2013 Oxford University Innovation Limited and subsequent improvements Copyright 2017-2024 by Oxford Semantic Technologies Limited.

This system is equipped with 8.5 GB of RAM, and RDOx is configured to use at most 7.7 GB (89.9% of the total).
Currently, 50.9 MB (0.6% of the amount allocated to RDOx) appear to be available on the system.
Since RDOx is a RAM-based system, its performance can suffer when other running processes use a lot of memory.

Access control has been initialized by creating the first role with name "guest".
This copy of RDOx is licensed for Developer use to Tom Vout (tom.vout@oxfordsemantic.tech) of OST until 24-May-2025 01:00:00.

A new server connection was opened as role 'guest' and stored with name 'scl'.
A new data store 'f1' was created and initialized.
Data store connection 'f1' is active.
Adding data in file 'data/prefixes.ttl'.
Prefixes will be updated after all inputs are processed.
Import operation took 0.004 s.
Warnings: no facts, rules or axioms were found in the specified source or sources.
output = "out"
The REST endpoint was successfully started at port number/service name 12110 with 8 threads.
WARNING: The RDOx endpoint is running with no transport layer security (TLS). This could allow attackers to steal
information including role passwords or authorization tokens. See the endpoint.channel variable and related
variables in the description of the RDOx endpoint for details of how to set up TLS.

> |
```

If not, see next slide
for common fixes

What's in the script?

```
start.rdfox
1  # Create a data store
2  dstore create f1
3  active f1
4
5  # Import prefixes
6  import +p data/prefixes.ttl
7
8  # Output results to the terminal
9  set output out
10
11 # Start the endpoint
12 endpoint start
13
```

You can execute these commands
one-by-one in the shell once RDOx is
running to achieve the same results.

Common fixes



File path needs quotes and/or backslash

All systems

command not found *is not recognised* *permission denied* *is a directory*

Problem: Some terminals handle slashes and special characters differently

Fix: Surround the file path (excluding 'sandbox') with quotation marks "" and, on Windows, replace forward-slashes '/' with back-slashes '\'

Unzipping creates a nested folder

Predominantly Windows

The system cannot find the path specified. *The term is not recognized*

Problem: Unzipping a folder sometimes puts the contents in a folder of its own name

Fix: Use the inner 'CDL24-RDFox-Workshop' as the working directory (open VS code at the inner folder)

Check your folder structure

All systems

Ensure your folder structure follows CDL25-RDFox-Workshop > RDFox-macOS-arm64-7.4b > RDFox.lic

Check your working directory and spelling

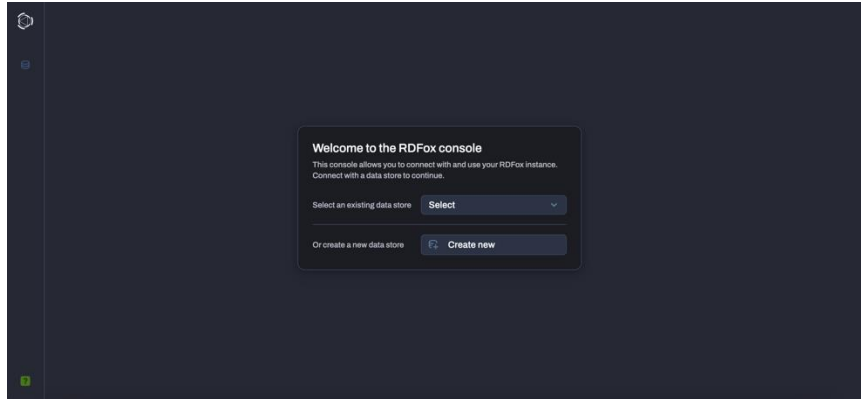
All systems

Ensure your terminal is running from the CDL25-RDFox-Workshop folder and your commands are correct

Open the RDFox Console

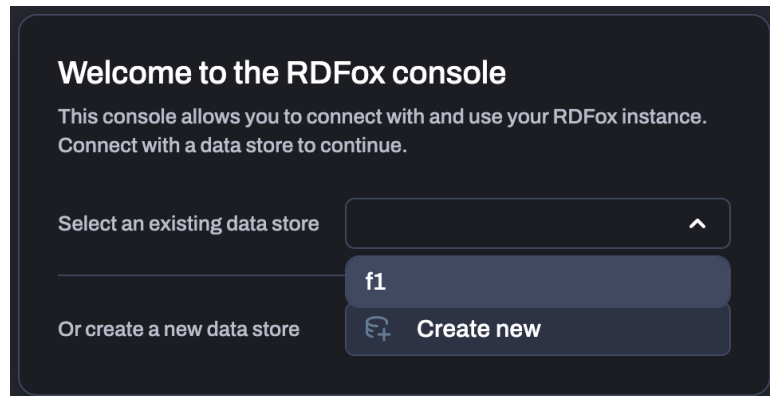


- 1 Open a browser and go to `localhost:12110/console`



[Click to open the console.](#)

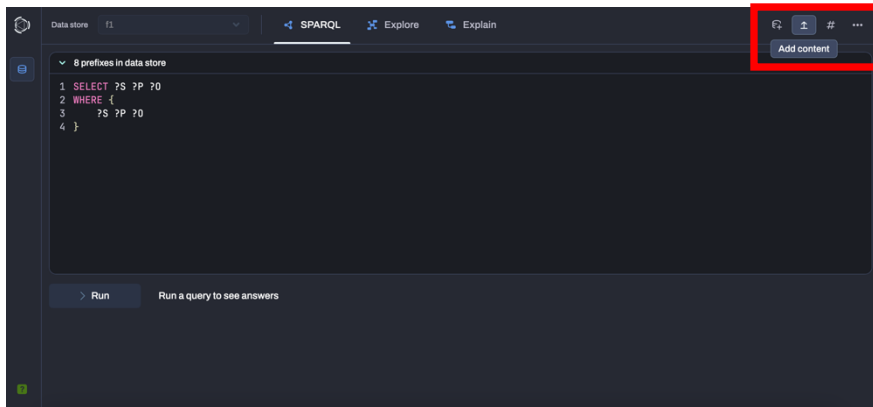
- 2 Select the 'f1' data store



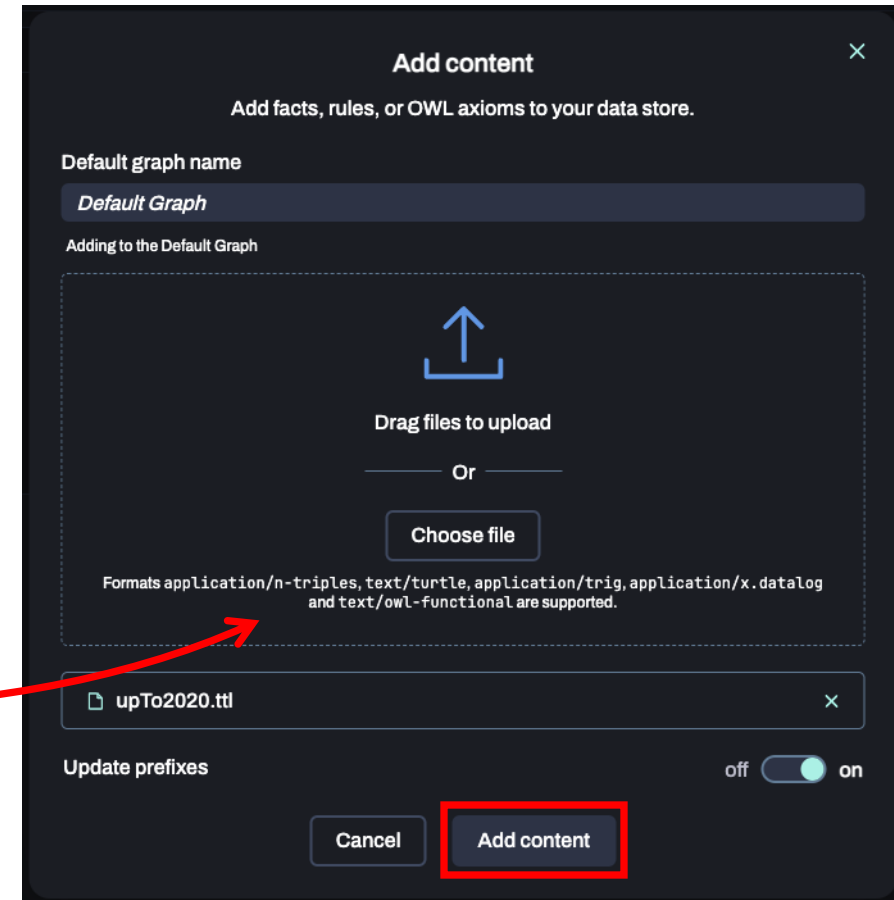
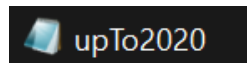
Adding data to RDFox



- 1 Click 'Add content' in the top right



- 2 Add 'upTo2020.ttl' from the 'data' folder





The data for today's class

Formula 1 data from the fan-run Ergast database

- Drivers
- Race results
- Races
- Constructors

Ergast Developer API

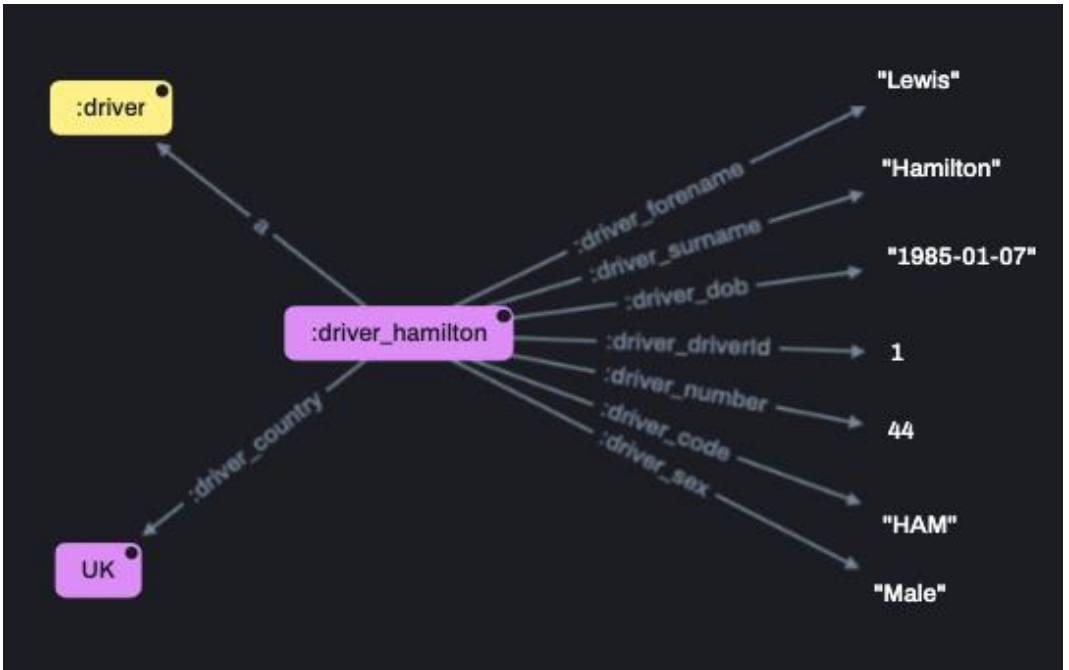
Query Results

Query Details

Series Page Results
f1 1 of 86 857

Driver Table

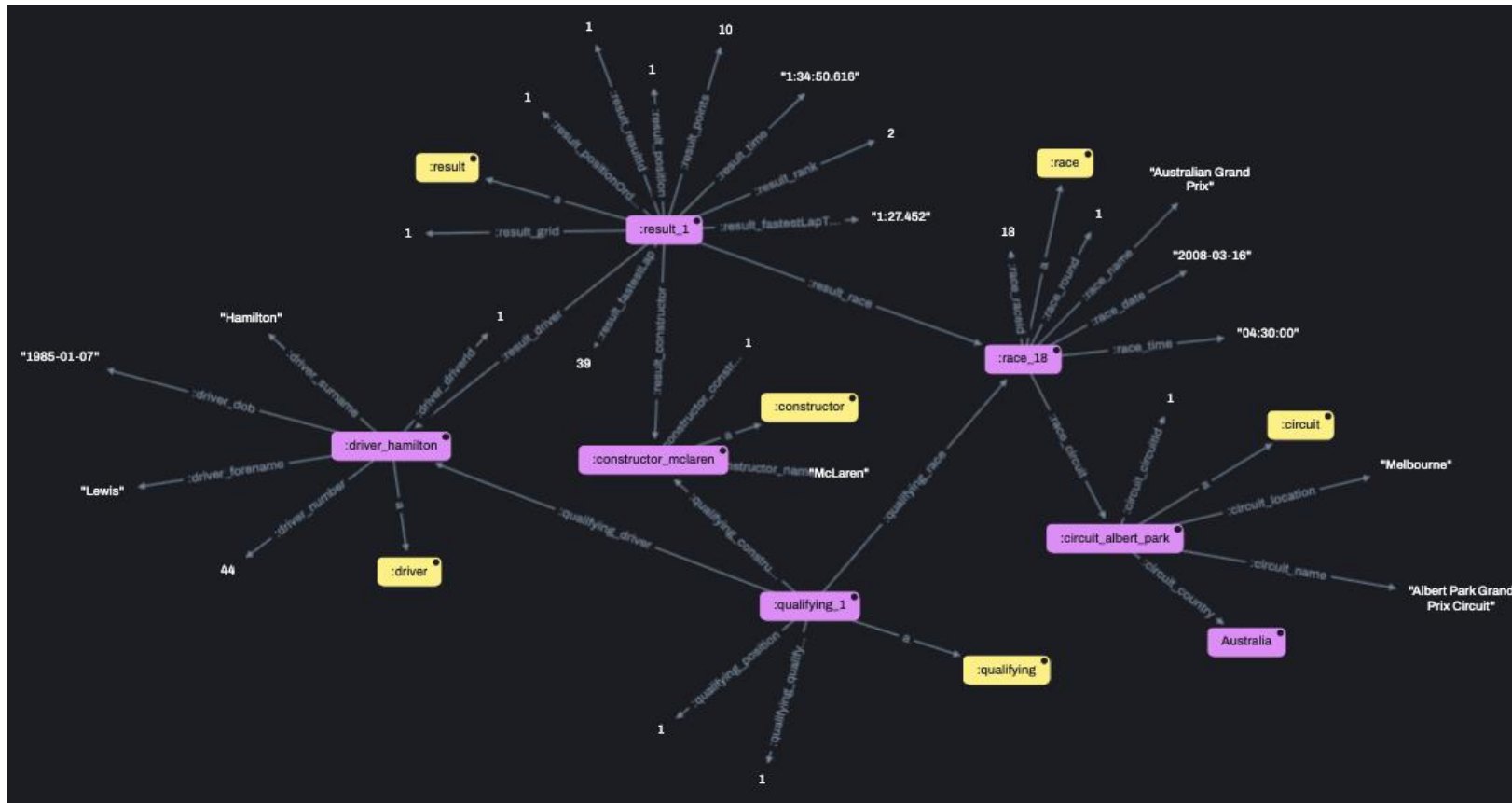
Driver Name	Permanent Number	Nationality	DOB	Information
Carlo Abate		Italian	1932-07-10	Biography
George Abecassis		British	1913-03-21	Biography
Kenny Acheson		British	1957-11-27	Biography



Exploring the data



[Click to open the console.](#)



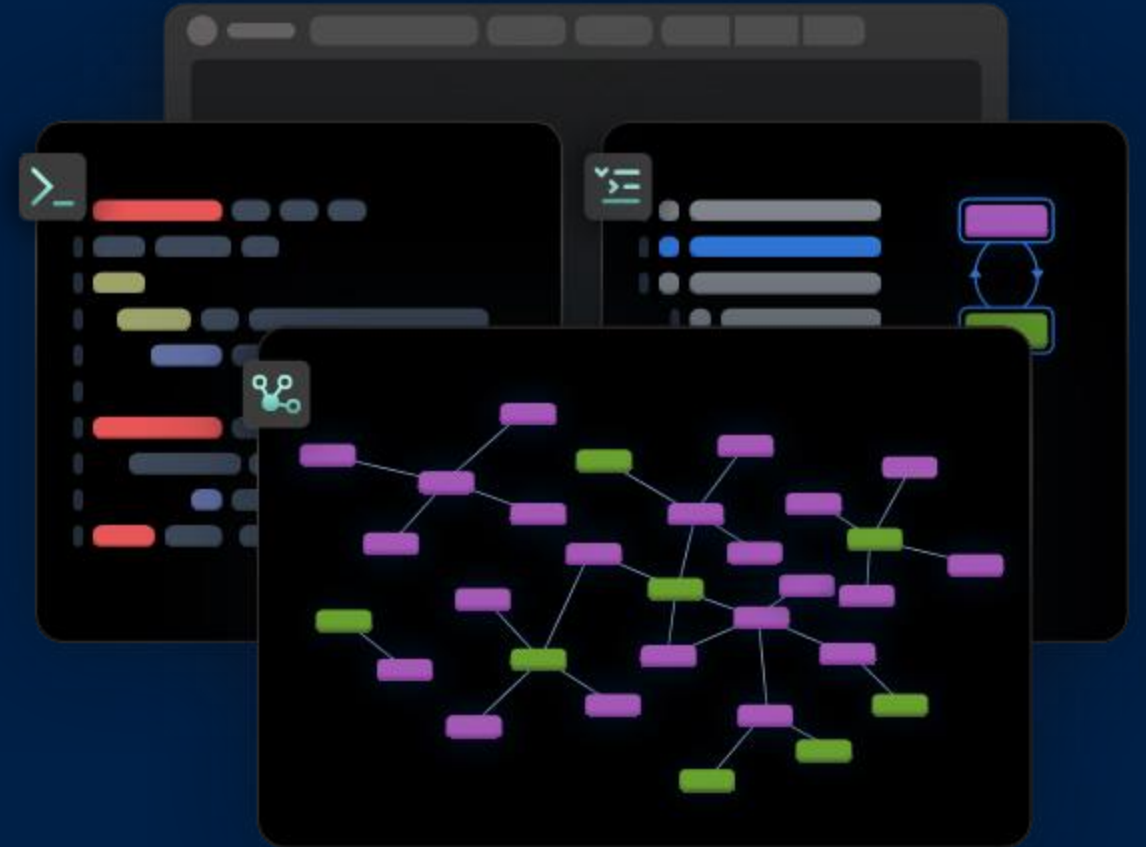
The graph shows a small sample of the data.

Classes are green,
instances of those
classes are purple, and
properties of those
instances have no
boxes.



SPARQL Querying

- Simple queries
- Classes and properties
- Aggregates
- Inner queries





Q1: Basic SELECT

Select queries are used to extract results from the dataset and can be modified to match whatever pattern you desire.

This is the simplest **SELECT** query. It returns all the triples in your dataset.

The **SELECT** keyword determines the type of query.

SELECT simply returns values.

```
1 # Query 1
2
3 # This query matches all the triples,
4 # and returns all the variables from the matched results
5 SELECT ?S ?P ?O
6 WHERE {
7     ?S ?P ?O
8 }
```

The output variables **?S ?P** and **?O** tell the **SELECT** query what to return from the variables that are found by the **WHERE** clause.

Here we're asking for everything.

The **WHERE** clause describes what the query is looking for.

In this case we want all triples, so we write the variables **?S ?P** and **?O**, showing we want cases where a subject, predicate, and object exist, with any value in any position.

We can name these variable anything we choose.

We still have to tell the query that the variable **?P** should come from the predicate of a triple, and that we want to consider all triples. Therefore, we still need **?S ?P ?O** inside the **WHERE** clause.

Q2: Counting Drivers' Races



→ Find the total number of races each driver individually has taken part in.

The **GROUP BY** clause states how the results should be grouped when returned.

Aggregate functions, such as **COUNT**, will now be performed for each group.

```
1  # Query 2
2
3  # Number of races per driver, ordered
4  SELECT ?driver (COUNT(?race) AS ?raceCount)
5  WHERE {
6    ?result :result_driver ___ ;
7    ?result :result_race ?race .
8  }
9  GROUP BY ___
10 ORDER BY DESC(___)
```

The **COUNT** function counts the number of instances of a variable, **?race**, and binds the resulting value as another variable, **?raceCount**.

The **ORDER BY** clause determines the order of the query results.

The **DESC** modifier tells the **ORDER BY** clause the variable indicated should be ranked in descending order.

Q3: Calculating Win Percentage



➔ Calculates the win percentages of all drivers.

Here we have used a query within a query, an 'inner query', or 'subquery'.

The **OPTIONAL** keyword states that if, for a given case, the subsequent result cannot be evaluated, skip over it and leave it undefined.

The **BIND** function sets a variable to a specific value.

The **COALESCE** function returns the first value in its list that does not throw up an error (including undefined).

```
1  # Query 3
2
3  SELECT ?forename ?surname ?raceCount ?raceWinsFinal ?percentage
4  WHERE {
5    # First get the drivers
6    ?driver :driver_forename ?forename ;
7            :driver_surname ?surname .
8
9    # First get the race count for each driver.
10   {SELECT ?driver (COUNT(?race) AS ?raceCount)
11     WHERE {
12       ?result :result_driver ?driver ;
13              :result_race ?race .
14     }
15   } GROUP BY ?driver}
16
17   # Then *optionally* get the win count for each driver.
18   OPTIONAL {SELECT ?driver (COUNT(?race) AS ?raceWins)
19     WHERE {
20       ?result :result_driver ?driver ;
21              :result_race ?race ;
22              :result_positionOrder 1 .
23     }
24   } GROUP BY ?driver}
25
26   # Using the COALESCE keyword, we make sure that when the number of wins is undefined
27   # it is 'bound' as 0.
28   BIND(COALESCE(____,0) AS ?raceWinsFinal)
29   # And use the two aggregate variables to compute a percentage
30   BIND(____ / ?raceCount AS ?percentage)
31
32 }
33 ORDER BY DESC(____)
```


Objectives



The Foundations

- ✓ What is Semantic Reasoning?
- ✓ What is RDFox?
- ✓ Setting up RDFox
- ✓ Querying with SPARQL

OWL Reasoning

- Why do we need reasoning?
- Sub & Inverse Properties
- Class Equivalence
- Unions & Intersections
- Limitations of OWL

Datalog Reasoning

- Basic rules
- Filters
- Aggregates
- Negation
- Binds
- Incremental Reasoning
- Materialization vs. query rewriting
- RDFox in enterprise



OWL Reasoning

- Why do we need reasoning?
- Sub Properties
- Inverse Properties
- Class Equivalence
- Unions
- Class Intersections
- Limitations of OWL



Web Ontology Language (OWL)

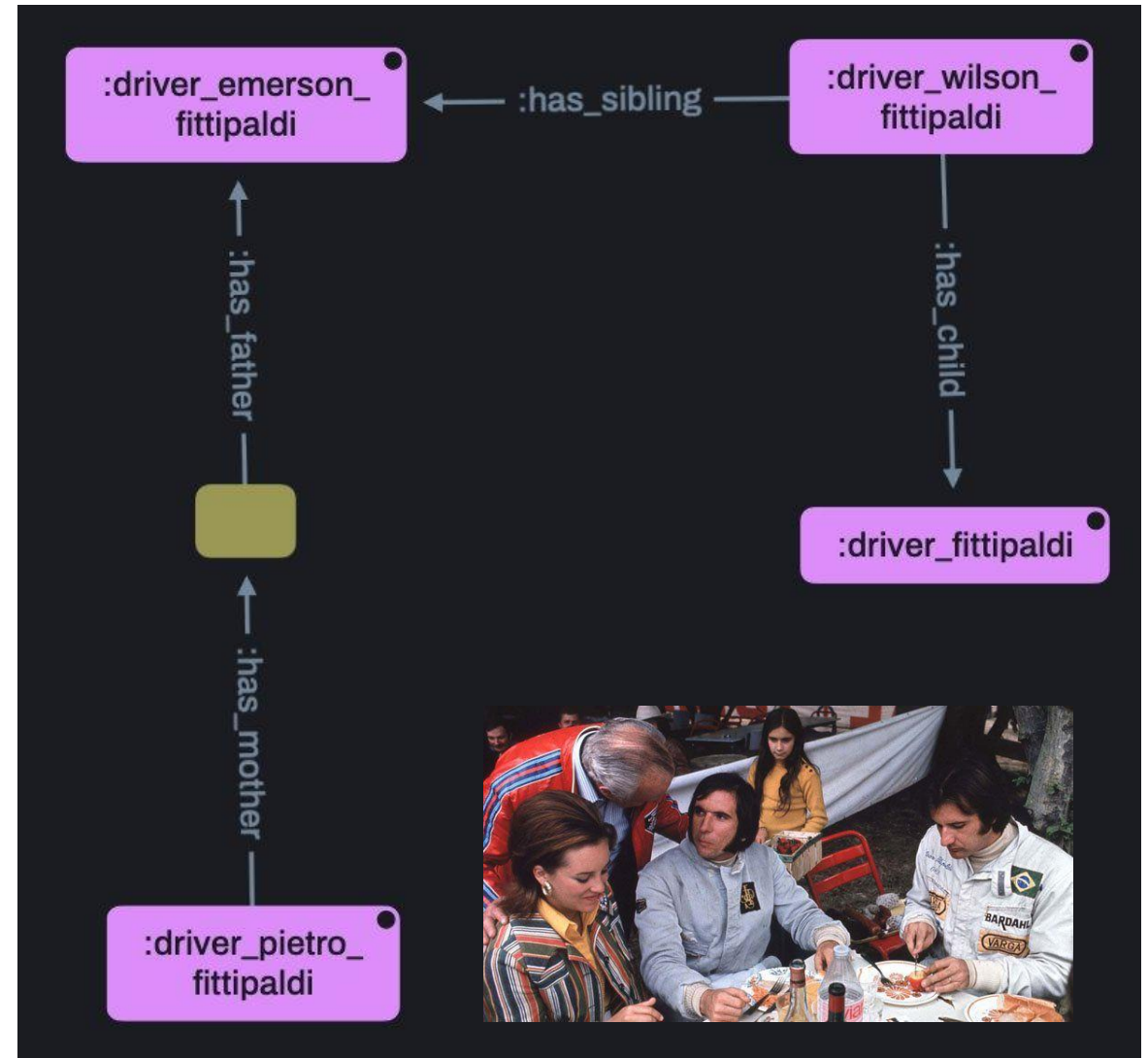


- **Standard created by W3C**, RDFox supports the profile OWL 2 RL
- Axioms are assertions about classes, properties or individuals
- Ontology are collection of axioms
- Ontologies are themselves graphs and can be written in many different syntaxes – we will use Turtle

Meet the Fittipaldi

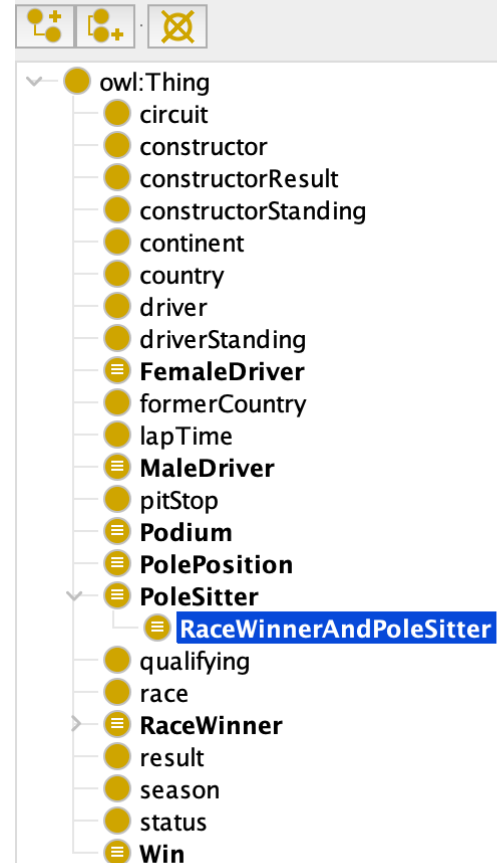


- Data is often irregular
- Is there something that can help us make it regular?
- As it happens, there is...



- Useful program for managing ontologies
- Free and open source
- Before writing an ontology, it is often better to have some data first and to consider the queries you want to facilitate

Class hierarchy: RaceWinnerAndPoleSitter





Axiom Set 1

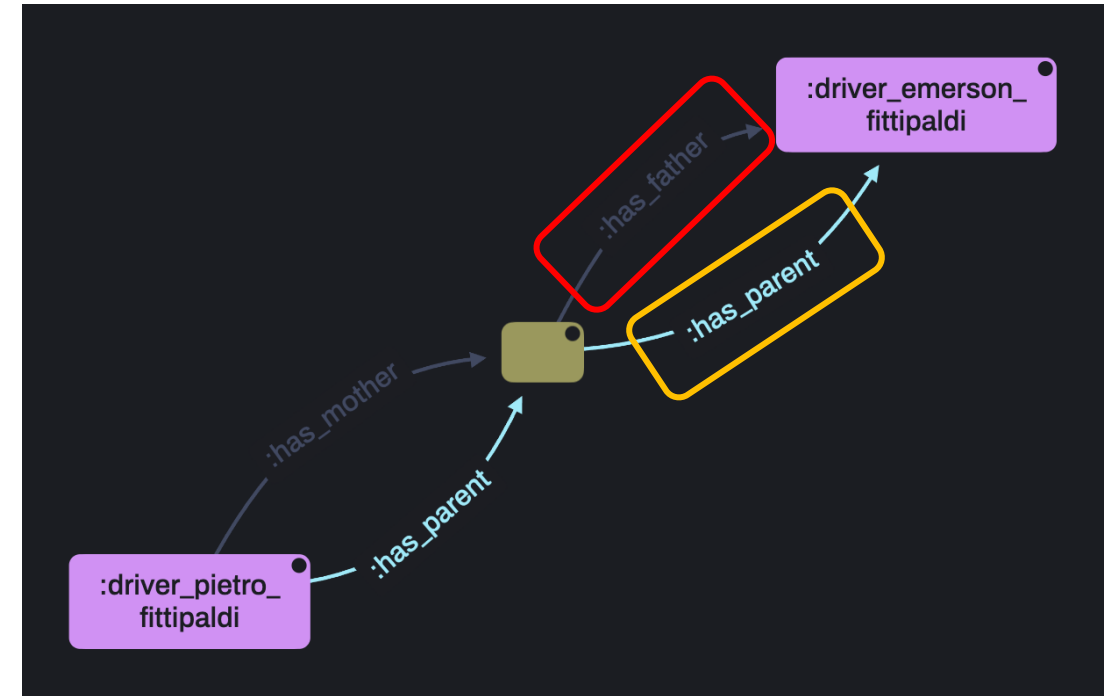
Sub Properties

'Father' and 'mother' relationships are sub-properties of 'parent'.

```
:has_father a owl:ObjectProperty ;  
rdfs:subPropertyOf :has_parent .
```

```
:has_mother a owl:ObjectProperty ;  
rdfs:subPropertyOf :has_parent .
```

'**:has_parent**' will be added between all nodes that share the '**:has_father**' relationship.



Multiple axioms can refer to the same properties.

'**:has_parent**' will also be inferred wherever nodes share '**:has_mother**'.



Axiom Set 2

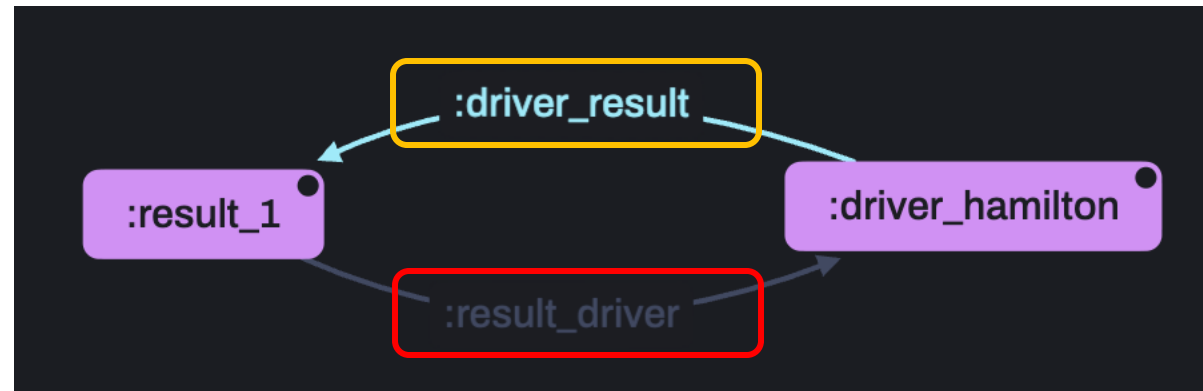
Inverse Properties

result_driver is the opposite of driver_result.

```
:result_driver rdf:type owl:ObjectProperty ;  
  owl:inverseOf :driver_result .
```

Only **Object Properties** can have **inverse** relationships as the subject of a triple must always be a node.

‘:driver_result’ will be inferred between nodes that share the ‘:result_driver’ relationship in the opposite direction, **swapping around the subject and object nodes**.





Axiom Set 3

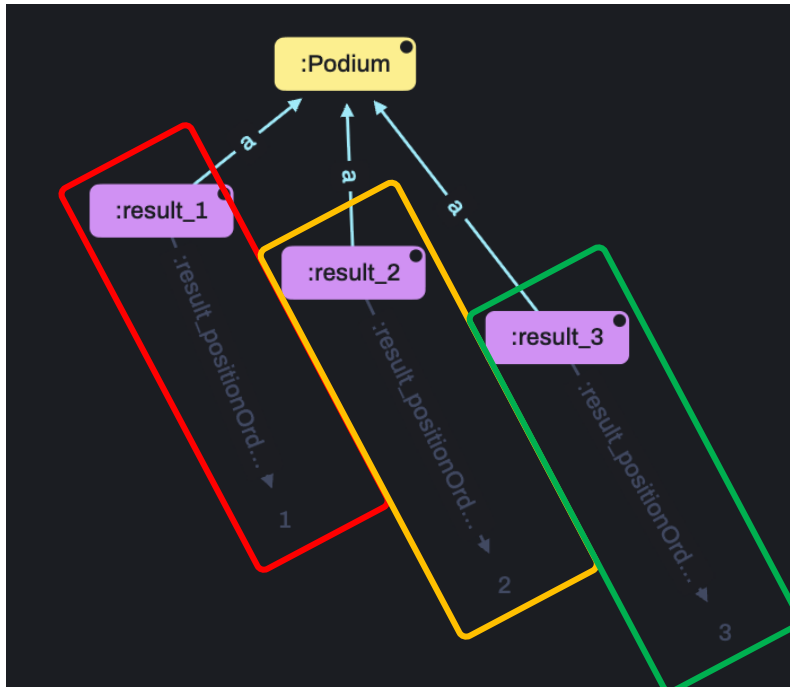
Unions

Accepting one thing OR another.

By using **owl:unionOf** we can accept several patterns.

In F1, a '**Podium**' means finishing **1st**, **2nd**, or **3rd**.

We can combine both **lists '()'** and **blank nodes '[]'**.



```
:Podium a owl:Class ;  
  owl:equivalentClass [  
    a owl:Class ;  
    owl:unionOf (  
      a owl:Restriction ;  
        owl:onProperty :result_positionOrder ;  
        owl:hasValue "1"^^xsd:integer  
      ]  
      a owl:Restriction ;  
        owl:onProperty :result_positionOrder ;  
        owl:hasValue "2"^^xsd:integer  
      ]  
      a owl:Restriction ;  
        owl:onProperty :result_positionOrder ;  
        owl:hasValue "3"^^xsd:integer  
      ]  
    )  
  ] .
```

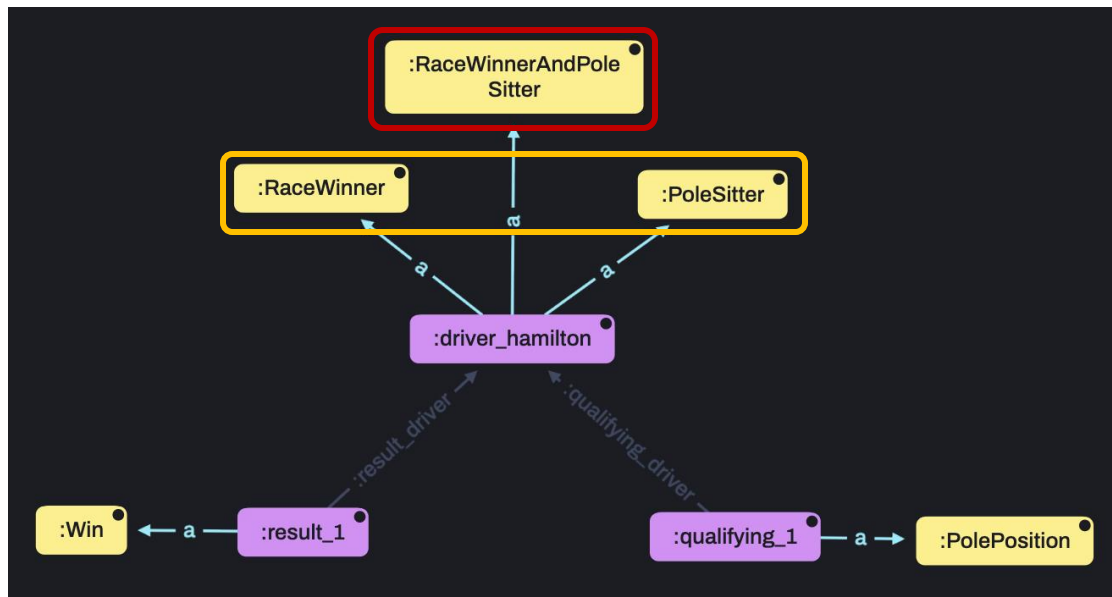


Axiom Set 4

Class Intersections

Combining one thing AND another.

Everything inside **owl:intersectionOf** must be included in the matched pattern.



```
:RaceWinnerAndPoleSitter
a owl:Class ;
    owl:equivalentClass [
        a owl:Class ;
        owl:intersectionOf (
            :RaceWinner
            :PoleSitter
        )
    ] .
```

Parsing the Ontology



Now we just need to import the axioms into a named graph, and then parse the graph for axioms:

```
import > :myAxioms axioms/axioms.ttl
```

```
import axioms :myAxioms
```

OWL 2 RL Limitations



OWL 2 RL does not support *existentials*.

This is why we see some warnings when parsing the axioms.

When it is known that a result is a podium, we have no way of knowing whether its finishing position is 1, 2 or 3, so we can't infer anything.

Similarly, when declaring a race-winner and pole-sitter we cannot specify that this must be from the same race.

RDFOx will still parse these axioms and facts will still be inferred where possible (if a result has position 1, it will be inferred to be a podium but not vice versa).



Removing Axioms

We will replace these axioms with Datalog rules, so we first need to remove them from our data store:

```
import axioms :myAxioms -
```

```
update ! clear graph :myAxioms
```

Objectives



The Foundations

- ✓ What is Semantic Reasoning?
- ✓ What is RDFS?
- ✓ Setting up RDFS
- ✓ Querying with SPARQL

OWL Reasoning

- ✓ Why do we need reasoning?
- ✓ Sub & Inverse Properties
- ✓ Class Equivalence
- ✓ Unions & Intersections
- ✓ Limitations of OWL

Datalog Reasoning

- Basic rules
- Filters
- Aggregates
- Negation
- Binds
- Incremental Reasoning
- Materialization vs. query rewriting
- RDFS in enterprise



Datalog Reasoning with RDFox

- Basic rules
- Filters
- Aggregates
- Negation
- Binds
- Incremental reasoning
- Materialization vs. query rewriting
- RDFox in enterprise



Datalog Rules



[*head facts*] :- [*body facts*] .

The ***head*** is ***inferred***
where the ***body*** is ***found***.



Rule 1

Basic rules

```
[?driver, :hasRacedIn, ?race] :-  
    [?result, :result_race, ?race],  
    [?result, :result_driver, ?driver] .
```

- This rule will add a direct link between drivers and the races they raced in. Notice this has the same effect as axiom 4, but expressed more clearly.

```
import rules/r1.dlog
```

- The same could seemingly be achieved with an equivalent SPARQL update, but rules offer several advantages over write queries.
- Check for inferred triples with query 11

```
evaluate queries/q11.rq
```

Rule 2



Filters

```
[?driver, :hasPodiumInRace, ?race] :-  
    [?result, :result_race, ____],  
    [?result, :result_driver, ____],  
    [?result, :result_positionOrder, ?positionOrder],  
    FILTER(?positionOrder < 4) .
```

- We can use **FILTERs** in rules too, in this case to find the races in which a driver reached the podium.

```
import rules/r2.dlog
```

- Check for inferred triples with query 12

```
evaluate queries/q12.rq
```

Rule 3



Aggregates

```
[?driver, :hasRaceCount, ____] :-  
    AGGREGATE (  
        [?driver, :hasRacedIn, ____]  
        ON ?driver  
        BIND COUNT(?race) AS ?raceCount  
    ) .
```

- This rule adds a count of races a driver has entered.
- Notice we use the previously inferred **:hasRacedIn** property. Rules can be composed together, and RDFox will automatically find the order in which to run them.

```
import rules/r3.dlog
```

- Check for inferred triples with query 13

```
evaluate queries/q13.rq
```

Rule 4



Aggregates

```
[?driver :hasRaceWinCount, ?raceWinCount] :-  
    AGGREGATE (  
        [?result, :result_driver, ____],  
        [?result, :result_positionOrder, 1] # Make sure the driver actually won  
        ON ?driver  
        BIND COUNT(____) AS ____  
    ) .
```

- This rule adds a count of races a driver has *won* (provided they won at least 1). We can put more than one atom *inside* the **AGGREGATE** statement.

```
import rules/r4.dlog
```

- Check for inferred triples with query 14, then evaluate query 2 again to see the performance gain.

```
evaluate queries/q14.rq
```

```
evaluate queries/q2.rq
```

Rule 5



Negation

```
[?driver, a, :DriverWithoutPodiums] :-  
    [?driver, a, :driver],  
    NOT EXISTS ?race IN (  
        [?driver, :hasPodiumInRace, ?race]  
    ) .
```

- This rule tells us that if a driver does not have a race where they were on the podium, then they are a **:DriverWithoutPodiums**.
- So, we are looking for something that is **not** in the data (i.e. no race where the driver was on the podium).

```
import rules/r5.dlog
```

- Check for inferred triples with query 15

```
evaluate queries/q15.rq
```

Negation as Failure



Negation as failure is negation understood as the *absence* of a triple.

Using Negation as Failure leads to ‘non-monotonic reasoning’- the engine may need to retract some triples instead of just adding them. This adds another dimension to an already complicated problem of reasoning planning, but RDFS handles all of that automatically.

Example What if a driver without any podiums so *far* gets a podium next year? In 2021, for instance, George Russell got his first podium. In our data, which is accurate up to and including 2020, George Russell would be a **:DriverWithoutPodiums**. But if we add data from 2021, we will need to retract this fact.

Exercise Run the query “russell.rq”, then import the data from file “2021-23.ttl”, and run it again.

```
evaluate queries/russell.rq
```

Rule 6



Binds

```
[?driver, :hasWinPercentage, ?percentage] :-  
    [?driver, :hasRaceCount, ____],  
    [?driver, :hasRaceWinCount, ____],  
    BIND(?raceWinCount/?raceCount AS ____).
```

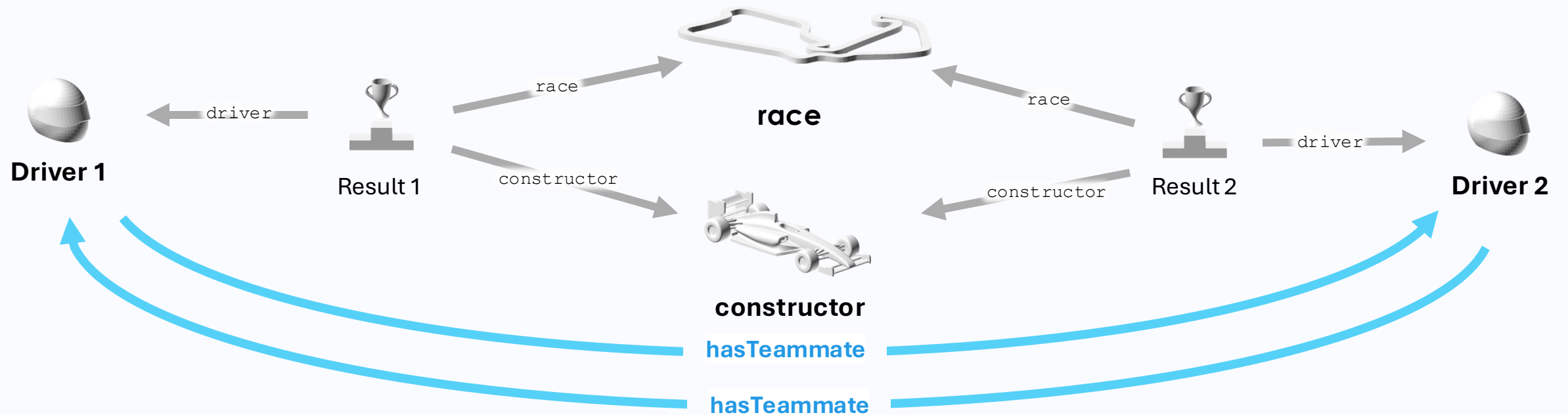
- With **BIND** we can use various mathematical functions, string manipulation, regular expression matching, conditional binds, hashing and IRI creation.

```
import rules/r6.dlog
```

- Check for inferred triples with query 16

```
evaluate queries/q16.rq
```


Bonus Exercise



Write and import a rule to find out if two drivers have been teammates (use the predicate **:hasTeammate**).



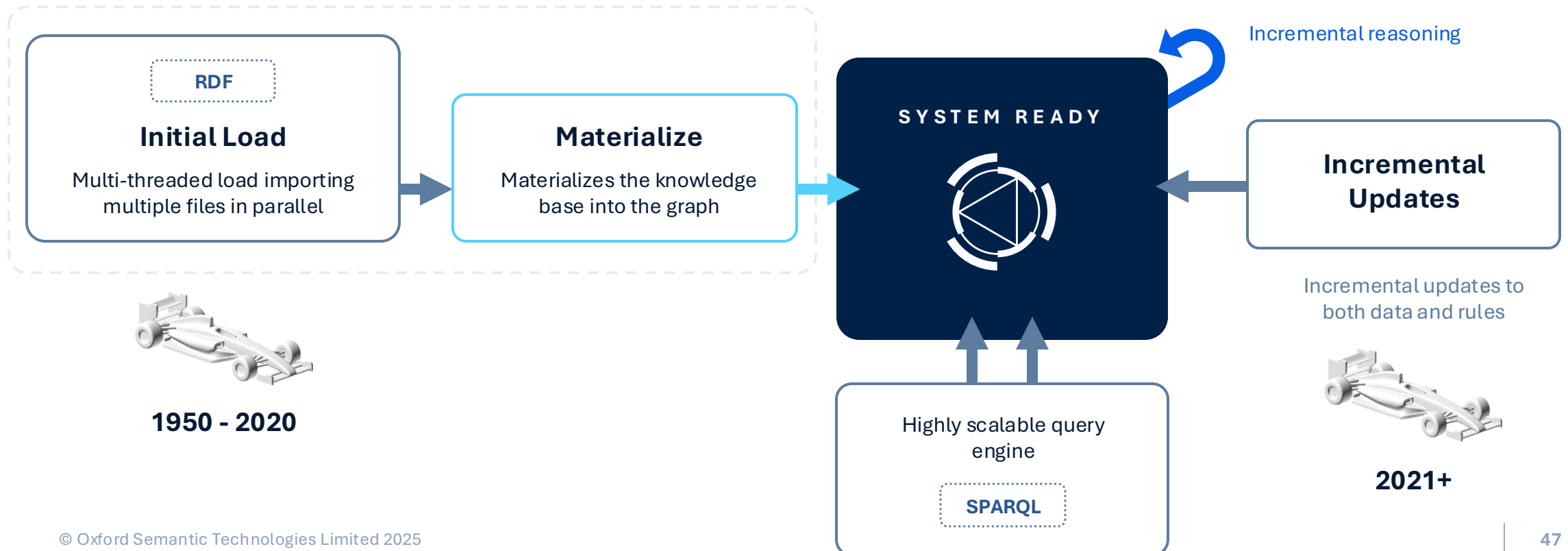
RDFox in Enterprise



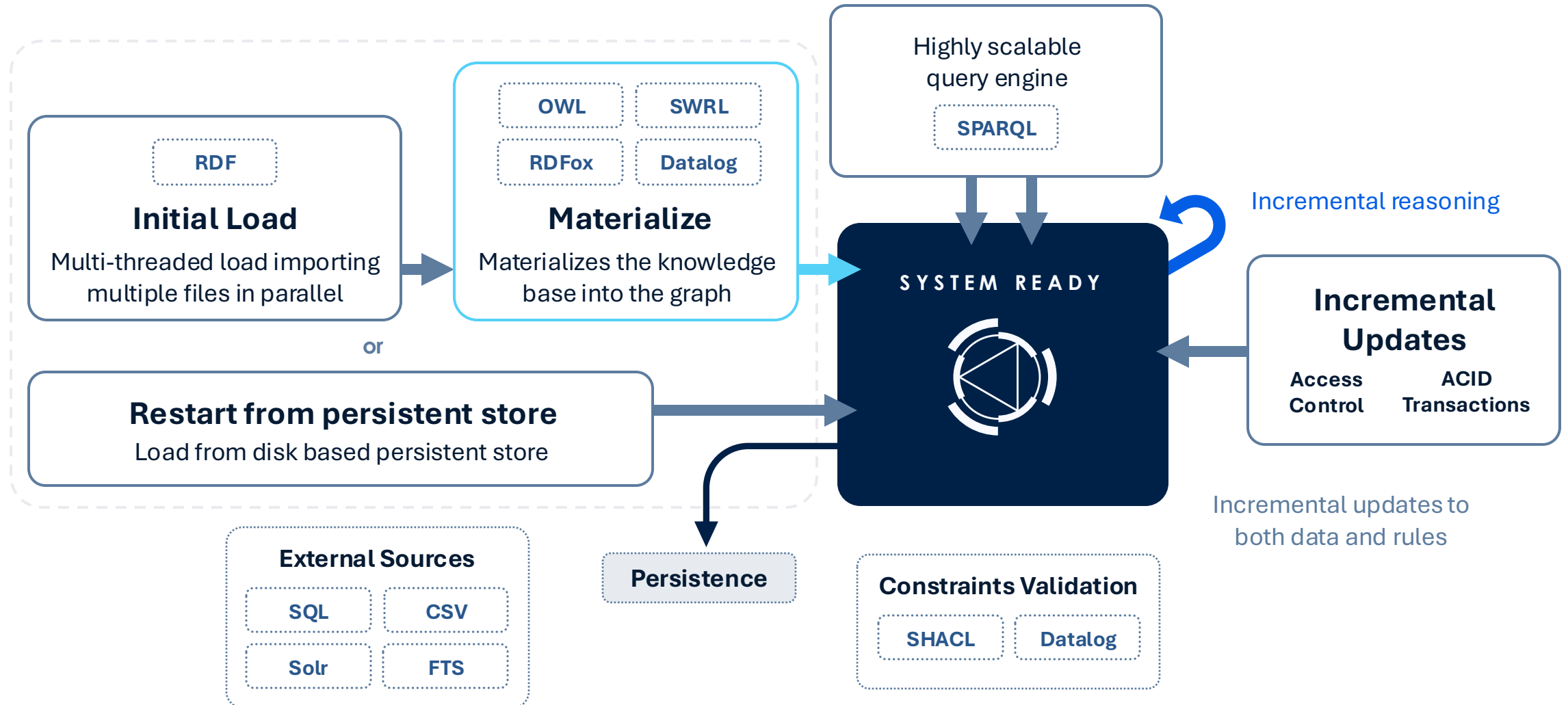
Incremental Reasoning



- Data can be imported at any time, and RDFox will automatically compute the necessary inferences.
- This all happens incrementally, without the need to reboot.
- Incremental reasoning opens the door to many novel use cases which were previously impractical.



RDFox : an enterprise solution



Deployment Options



**Mobile & edge
devices**



**Personal
machines on
premises to cloud**



RDFox vs. other solutions



Feature	Example	RDFox	Materialisation Competitors	Query Rewriting Competitors
Ontological reasoning	List all financial instruments / derivative instruments / futures. (Class inferencing)	✓	✓ *	✓ **
Graph analytics	What is the nearest electrical switch for a circuit? (Negation & Recursion)	✓		
Data analytics	What is the total volume and value of trades? (Aggregation & Arithmetic)	✓		
Local constraints validation	No trades over a limit & no traders without trades! (Filters & Negation)	✓	Partial (SHACL)	
Global constraint validation	Every component must be connected to a power source! (Negation & Recursion)	✓		
All of the above	What is the total value of assets owned through holdings?	✓		

* many times slower than RDFox

** orders of magnitude slower than RDFox

Graph RAG with Reasoning



Photo by [ilgmyzin](#) on [Unsplash](#)

LLMs





-  Offer a powerful, flexible and accessible way to interact with data
-  LLMs often hallucinate and are a black box, meaning results cannot be explained





Photo by [Growtika](#) on [Unsplash](#)

RAG

-  Grounds answers in data
-  Significantly restricts range, flexibility and complexity of possible questions



RAG & Reasoning

-  Enriched data enables RAG to answer a broader range of questions, returning complex insights
-  Explainability means answers are auditable

Congratulations!



The Foundations

- ✓ What is Semantic Reasoning?
- ✓ What is RDFox?
- ✓ Setting up RDFox
- ✓ Querying with SPARQL



OWL Reasoning

- ✓ Why do we need reasoning?
- ✓ Sub & Inverse Properties
- ✓ Class Equivalence
- ✓ Unions & Intersections
- ✓ Limitations of OWL

Datalog Reasoning

- ✓ Basic rules
- ✓ Filters
- ✓ Aggregates
- ✓ Negation
- ✓ Binds
- ✓ Incremental Reasoning
- ✓ Materialization Vs. query rewriting
- ✓ RDFox in enterprise

Additional information



Visit our website

<https://www.oxfordsemantic.tech>



Our blog

<https://www.oxfordsemantic.tech/blog>



Read our documentation

<https://docs.oxfordsemantic.tech/index.html>



Request an evaluation license

<https://www.oxfordsemantic.tech/free-trial>